Predicting Popular Xbox games based on Search Queries of Users

Chinmoy Mandayam and Saahil Shenoy

I. Introduction

This project is based on a completed Kaggle competition. Our goal is to predict which Xbox game a user on the Best Buy site will be most interested in based on their search query. Essentially this will be a recommender system type problem, which we are looking to solve by building a multi-label classifier.

II. Data Description

Each training example has the following information:

User token	SKU clicked	search query	click time	query time

We have about 42,365 training examples. the come from over 38,000 users over almost 3 months from August-October 2011. In addition, we are given a separate list of about 870 Xbox 360 related SKU's and some meta information (including the product description, price history, etc.,) about these items. This list has multiple duplications, and after culling these, we find that this list contains 437 unique entries The test data has the same information as the training data except for the SKU, which needs to be predicted.

Performance evaluation

We decided to perform our testing using the same metric that the Kaggle challenge uses, MAP@5 (*Mean Average Precision* at rank 5). The challenge expects a list of the top 5 recommendations for the each search query. Then, the precision of the recommendation is computed (using the actual click information) for the set of the top n choices , for each n from 1 through 5. This is then averaged to obtain the *average precision* at rank 5(ap@5) score for the query. Effectively, the score for a particular recommendation is $^1/_k$ if our k-th recommendation matches the actual product that the user clicked on, and zero if the actual product was not recommended. The mean of the average precision values for all the queries gives us our MAP score.

In our our midterm report and our poster, our reported scores on Kaggle were significantly lower than our actual performance giving us an impression that our models had a high variance, or that the test data had a different distribution from the training data. However, this discrepancy was only because we had misinterpreted the file format required of our submissions, so only a portion of our predictions was actually being tested. We have corrected this error in this report

III. Classifier models

Since we are expected to give more than one choice, a simple classifier which gives us one recommendation without a good confidence measure would be a bad choice. We need a way to rank our recommendations. So, we must immediately rule out vanilla SVMs. A Bayesian model (such a naive Bayes classifier) would quite naturally have a probability interpretation that could be used to compare the likelihood that a particular product will be selected. We could also use logistic regression, which inherently attaches a sigmoid confidence measure to each classification, $g(\theta^T x)$. The closely related soft-max regression would have a similar interpretation. We decided to try out naive Bayes and logistic regression based classifiers.

IV. Classification Features

Since the number of search queries per user is so low, we decided to ignore the user information (which we could have run collaborative filtering on) and instead concentrate purely on the search queries and the time information. Text classifiers generally use a "bag of words" based model, where the number of occurrences of a each word in a

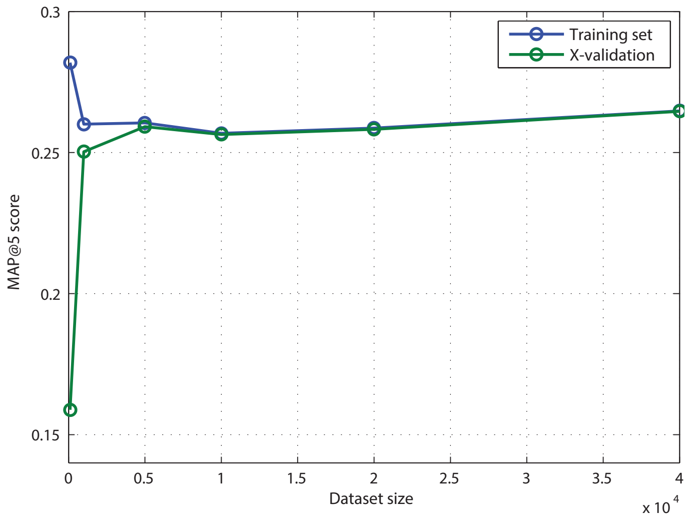

Figure 1. Cross-validation performance of multinomial naive Bayes

vocabulary is the feature vector for each sample. However, since our search queries are we decided to first try models based on a "bag of letters" model, where the feature vector is the number of time each character (including numerals) appears. Based on this simple feature vector, we tried using a naive Bayes classifier and a logistic regression based classifier. This feature vector was extracted using a Python pre-processing routine.

V. Multinomial Naive Bayes.

We trained a multinomial naive Bayes classifier in MATLAB, similar to the one we had in Homework 2. We then retrieve the 5 SKU's with highest probabilities. We performed 10-fold cross-validation on data-sets of up to 40000 training examples. As we can see from the graph, the results were quite disappointing, both in the training error, as well as in the performance over the test data set, which gave us a score of only 0.267 compared to the top score of 0.79. Therefore, although we seem to have a low variance, we have a problem with severe bias.

VI. Logistic regression and Soft-max regression.

Using the same feature vectors, we trained a one-versus-rest logistic regression classifier with L_2 regularization for each SKU that appears in the training data in MATLAB. We can then predict the classification probabilities $g(\theta_i^T x)$ for each SKU. We then retrieve the 5 SKU's with the highest $\theta_i^T x$ for each x, which is our recommendation. Unfortunately this algorithm is quite slow (each run with the full training data takes about 10 minutes), which prevented us from doing a full k-fold cross-validation analysis. When we used a small subset of the data (about 6400 samples) to train the classifier, we obtained a cross-validation error of 0.71. The performance on the Kaggle competition data was also a much more reasonable 0.71. However, training with a much larger data set of 36,000 samples did not improve the performance too much. Cross-validation gave us a score of 0.73 , and the performance on the Kaggle competition test data was actually slightly better, 0.740. Clearly, we still have a problem (albeit smaller than before) with model bias,.

We also tried performing logistic regression using the number of occurrences of ordered pairs of letters (e.g. 'aa','ab','ac',...) as well. This increased the feature vector size greatly, which is usually a good way of removing bias. However, this also increased the computational complexity of the problem. We tested a classifier trained on a relatively small sample set of about 6400 samples. We found that the improvement compared to using single letter frequencies was statistically insignificant, especially considering the enormous increase in computational requirements.

We also tried to add some features corresponding to the date of the search query. The data was collected over approximately three months. New video games are usually released on Tuesdays, with major titles being reasonably well spaced, similar to movie releases. Therefore the time of a query can also be helpful in predicting user interest in a game. The dates were first binned into one of 6 groups (about two weeks each) and assigned a feature vector with 7 components based on their position in the bin, using linear interpolation. This gave us a small improvement in the score to 0.745.

We also implemented a soft-max regression solver for the same feature set. A batch gradient decent based optimizer was built in MATLAB. This solver was computationally slow, requiring over 30 minutes to converge. The additional computation had no benefit however, with soft-max regression giving us almost the same performance as one-versus-rest logistic regression, with a score of 0.743.

We also attempted build SVMs for this classification problem. To build a multi-class SVM, we trained a one-vs-many classifiers for each product, as before. We used the functional margin as a heuristic measure of the classifier's confidence level. Unfortunately, this technique does work very well, and we could only get as score of 0.629 with a linear SVM and 0.622 using a Gaussian Kernel.

VII. Word Frequencies

After attempting supervised learning techniques on the training and test set with the single character (includes letter and numbers) frequencies, the next step was to change the feature space of all words used in all the queries. In other words creating a binary vector that is of the same size as the dictionary of words used in the queries from both the training and test set. The methods applied to this test set were the logistic regression and SVM. This turned out to perform poorer than in the same algorithms as was used before for the training and test sets that use the single character frequencies. A suspected reason that this didn't get a higher success rate is because the size of the feature space was much bigger which resulted in over fitting. One interesting thing that was found is that in both cases logistic regression did better than the SVM classifier. Results for each algorithm summarized in the table below:

Algorithm	Test Success
Logistic Regression	0.50951
SVM (with linear kernel)	0.3403

VIII. Clustering Products

Part of the data given is the product name with relevant information (such as date released, price, sales rank, etc.). The goal of clustering is to figure out which products are similar to one another. The end goal of doing this is to use this information in order to help with the classification aspect of this problem, by giving it slightly more accuracy about which class it should be looking at. A few pre-processing steps were taken before clustering was implemented. As a first step towards getting something to run columns were eliminated if they were either not filled in for each product, corresponded to a id number, or if all the values in the columns took on the same value for each product (which arose in the some of the Boolean valued columns where each entry for each product was labeled as false). Name of the product wasn't included since it varied a lot. There were a total of 437 products total and 121 features originally. After this pre-processing of removing irrelevant features (where irrelevant is defined by the criteria listed above), there are now a total of 28 features. All text features were converted into bit vectors where each feature of the bit vector was a word or sentence depending on the text feature. Once the pre-processing is done all rows (corresponding to each product) were standardized via subtracting the mean and dividing by the standard deviation for each row. Standardizing each row makes all the numbers of similar scale. Therefore a Euclidean distance norm can be used. To heuristically determine the number of clusters that should be used a spectral graph theory technique was used, in which the Laplacian matrix was diagonalized and the greatest difference in the each of consecutive eigenvalues was used [1]. Then normal k-means was used to cluster the objects. This clustering technique didn't not yield meaningful results as it did not group similar product names together (e.g. *Call of Duty* and *Call of Duty II*). So one modification was made in which products with similar names were automatically clustered together, which shall be denoted as pre-clustering. Pre-clustering now yielded 280 pre-clustered objects together. Now each row is the average of each product in the pre-clustered objects. The same spectral graph theory technique was used to figure out the number of clusters and kmeans should use on these 280 pre-clusters. The purpose of all this clustering is to cluster the product names together for the purpose of making the classification part of this project more accurate. There will be two levels of training happening. The classification algorithm (SVM or logistic regression) will train on knowing which cluster each example belongs to. Once that is done another SVM or logistic regression will be implemented within that cluster to know which of the products the user is interested in. In the end this didn't not make the over all algorithms perform better by doing this two-step supervised learning process via clustering.

IX. Language Processing techniques

The aforementioned techniques are generic tool, and do not make use of the structure inherent in search queries for games. There is a surprisingly large degree of similarity between search queries, as compared to even generic text. For instance, we

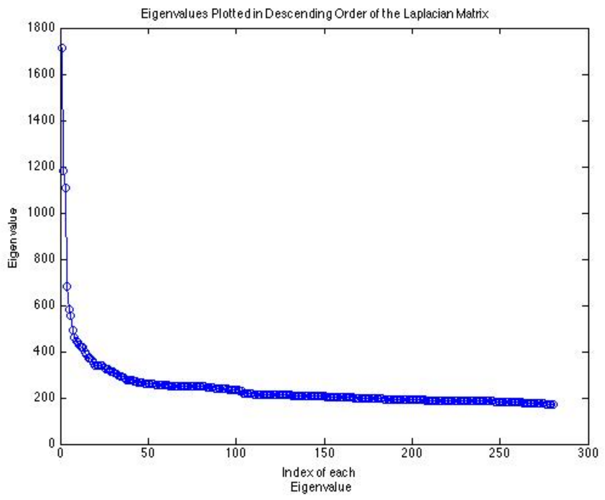

Figure 2. A graph of the eigenvalues of the Laplacian matrix plotted in descending order. Circles denote each of the eigenvalues. Using the index labeling given on the horizontal axis, the top 5 greatest differences occur between consecutive eigenvalues of 1 and 2, 3 and 4, 4 and 5, 2 and 3, and 6 and 7.

found that after performing some additional pre-processing (like separating words from numbers and eliminating spaces and other such non-informative characters, we found that there were only 3632 distinct word strings in the entire training data (even without accounting for word order). This naturally leads to the following idea: if each search query corresponded to a particular game, we'd only have to look in our training set for the same query string and thus find the corresponding product. Of course, this is not entirely true, since the same query could lead to different products (especially if they are just minor variants). So, we have to consider the *probability* that a particular query corresponds to a certain product. Thus we can use an approach like Naive Bayes (the multivariable version is especially suitable). In particular, each search query is characterized by just two features:

1) A concatenation of the words.
2) Any specific numbers that appear in the string (this helps distinguish between items like *"Battlefield 2" and "Battlefield 3").*

We built a naive Bayes classifier based on this feature set in Python. To our surprise this gave a score of 0.72, almost approaching the performance of our naive. This is even more remarkable considering that the computation for this classifier runs in less than 10 seconds.

Upon investigation of the cross-validation examples that the algorithm misclassified, we found some particularly interesting modes of failure:

1) A change in just 1-2 words in the search query caused the query to not match any training example
2) A search query for a game not searched for in the training set, but the full name of the product is present in the meta data file.

To tackle both these problems we use the following heuristic; if a query has no matches, we separate out the words in the query and compare this set of words with the same set for all the other queries in our training examples. We then replace this query with the query with highest number of matching words. We also add a small weighting factor to the match based on the frequency the query. We developed a scheme to efficiently compute the best matching word for each query. Essentially, this implements a k-nearest neighbor algorithm (we used k=1 for simplicity) for the queries in word-frequency space. Equipped with this function, we can now easily handle the second case as well, by just creating a dummy query based on the full-name of the product for each product in the meta-data file.

In addition we also added the date based feature discussed in the section on Logistic regression. Using all these features together in our Naive Bayes framework, we obtained our best score of 0.753 on the Kaggle private leaderboard. The whole algorithm takes less than 40 seconds to complete. A graph of k-fold cross-validation performance for this algorithm is shown in Figure 2. We see that the cross-validation error does not converge to the self-validation error, a sign of lingering variance. While this can often lead to over-fitting, in this case, a fundamental reason for the difference is that during self-validation, we have already observed all the queries that we are testing. When we perform cross-validation, new variants of search queries can and do appear . Since no model exists for these new variants, we are not over-fitting in the classical sense.

X. Conclusions

We have implemented several algorithms and heuristics to try and tame this massively multi-class classification problem. We found that using general statistical learning tools, such as logistic regression

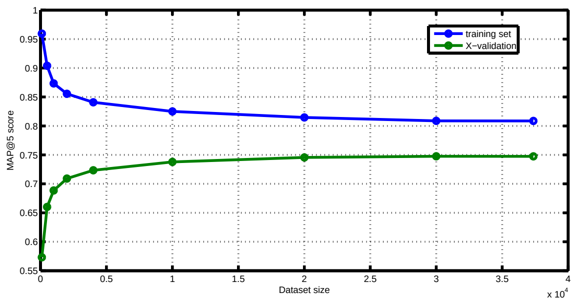

Figure 3. Cross-validation performance of multivariable naive Bayes

and soft-max regression can perform very well in multi-class classification problems. However, using the margin as a measure of confidence for SVMs is a bad idea, and it is advisable to perform some probability fitting as suggested in [5] before performing multi-class classification with on one-verses-rest SVM classifiers. We also observed that some very simple and computationally fast learning techniques, when coupled with a knowledge of the structure of the data, can actually outperform theses generic algorithms.

Our final score of 0.753 (the highest score was 0.789) on the MAP@5 metric would have ranked us 19th among the 97 teams that participated in the original Kaggle challenge. We believe, further improvements, such as adding a spelling correction module can help us improve the performance of our scheme further.

XI. Acknowledgments

Would like to give a big thanks to Nick Hahner, Allison Carlisle, and Paola Castro for providing a program written in python using Scikit-learn to run some of these supervised learning algorithms including SVMs .

References

[1] *Data Mining Hackathon on (20 mb) Best Buy mobile web site* - ACM SF Bay Area Chapter http://www.kaggle.com/c/acm-sf-chapter-hackathon-small/

[2] *Mean Average Precision* https://www.kaggle.com/wiki/MeanAveragePrecision CS229 Lecture Notes http://cs229.stanford.edu/materials/

[3] *Soft-max Regression* http://ufldl.stanford.edu/wiki/index.php/Soft-max_Regression

[4] *CS246 Lecture Notes – Discovering Clusters in Graphs: Spectral Clustering* http://www.stanford.edu/class/cs246/slides/12-spectral_ml.pdf

[5] Platt, John. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods." *Advances in large margin classifiers* 10.3 (1999): 61-74.

[6] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.